Technische Universität München
Lehrstuhl Informatik V
M. Bader, A. Pöppl

WS 2018/19
Worksheet 1
October 31st, 2018

Tutorial: HPC - Algorithms and Applications WS 18/19

Complete the following assignments (alone or in a group), and send *only* your source code via e-mail to `poeppl@in.tum.de` until Sunday, November 11th, 2018.

Worksheet 1: Matrix Multiplication in CUDA

T1.1: Basic matrix multiplication

a) Write a simple matrix-matrix multiplication for small matrices (up to $n = 16$). The following tasks are necessary:
   - Compute the amount of memory required for storing an $n \times n$ matrix with single floating point precision.
   - Allocate device memory, transfer the input matrices **A** and **B** from host to device memory and the result matrix **C** back to host memory, deallocate device memory.
   - Define grid and block size and call the device kernel. You can assume for now, that a single block is sufficient for a matrix.
   - Implement a basic matrix multiplication kernel in the function `matrixMultKernel_global`

T1.2: Increase of problem size

a) Extend the code by allowing matrices of size $n > 16$.
   - Change your grid and block size computation to handle $n \times n$ matrices for any $n > 0$. You can assume that the matrices fit into device memory.
   - Change your matrix multiplication kernel to handle the new grid and block sizes.

b) Measure the execution time using different block sizes. Compare the execution time with the execution time of a CPU code by replacing the call to `CUDA_matrixMult` with `CPU_matrixMult`. Find the optimal block size for a matrix of size 256×256.

H1.1: Make it run

Assignments a) and b) are mutually exclusive, complete *only one of them*.

a) Compile the exercise code on a local machine:

- Verify you have a CUDA-capable GPU and install the CUDA toolkit.
- Download and extract the exercise code from `http://www5.in.tum.de/wiki/index.php/HPC_-_Algorithms_and_Applications_-_Winter_15`, i.e. into `~/HPC/Exercise1`
- You might have to export some path variables:
  `export PATH=$PATH:<cuda_dir>/bin`
  `export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:<cuda_dir>/lib64`
- Change to the folder `cd ~/HPC/Exercise1` and type `make`, compilation should work.
- Execute the program using `./out <MATRIX-SIZE> <NUM-REPEATS>`

b) Compile the exercise code on the SCCS cluster:

- Know how to access the cluster:
  - **Host:** pproc-be.in.tum.de
  - **User:** <Your TUMOnline ID>
  - **Password:** <Your TUMOnline Password>
- Copy the necessary files to the cluster (using `rsync`, `scp`, or any SCP tool of your choice)
- Open an SSH connection to the cluster, then open an SSH connection into one of the GPU nodes **gpunode<1-4>**.
- Change to the folder you copied, and type `make`. Compilation should work.
- Execute the program using `./out <MATRIX-SIZE> <NUM-REPEATS>`

H1.2: Tiled matrix multiplication

a) Implement a tiled matrix multiplication kernel for improved memory performance. The tile size is defined in a preprocessor macro called `TILE_SIZE`.

- In the function `matrixMultKernel_tiled`, allocate shared memory that holds matrix tiles of size `TILE_SIZE` $\times$ `TILE_SIZE` for the matrices **A** and **B**.
- Fill the shared tiles with data from the matrices.
- Perform a partial matrix multiplication on the shared tiles.
- Set appropriate thread barriers in order to sychronize all threads of a block.

b) Compare the performance of CPU, basic GPU and tiled GPU matrix multiplication for a matrix of size 256×256.

- Which implementation is the fastest?
- Are there differences for different values for `TILE_SIZE`?